Department of Electrical Engineering

The Cooper Union for the Advancement of Science & Art

SpellCorrector

Spell Correction of Text from the Tesseract OCR Engine

by

Nick DeTommaso, Elizabeth Kilson, Collin Stocks, and Bobby Yankou

Advisor: Prof. Rob Marano

ECE465: Cloud Computing

Spring 2014

Contents

List of Figures

1 Summary

With the advent of the internet, one goal has been to digitize printed books so that they may be made easily available to anyone with an internet connection. Many initiatives have been created with the purpose of digitizing as many books as possible for easy access online. One such project, the Google Books Library Project, which began in 2004–2005, works with over 40 libraries around the world to digitize their collections. According to the New York Times, the Google Books Library Project has scanned over 20 million books, as of November 2013[2]. To make these books searchable, Google and other databases of scanned books need to be able to identify the words that appear in the books. When analog texts are digitized, pages are scanned, creating image files. Since it is desirable to make the texts searchable, text files must be created from the image files. Tesseract is an Optical Character Recognition (OCR) engine that receives images and outputs text interpreted from the images. Since Tesseract is not 100% accurate, the interpreted text can be improved using Natural Language Processing. According to Google Books Search, there are almost 130 million unique books in the world, meaning they have 110 million books left to digitize[3]. For such a large volume of text, it would be advantageous to employ a distributed spell corrector to improve the accuracy of the interpreted text.

For the present spell correction scheme, a Bigram Language Model uses the probability of two words appearing in sequence to correct the spelling of words output by Tesseract. Without distributing, this process takes an inordinately long period of time. For a particular test file—specifically, one tenth of Jane Austen's *Pride and Prejudice*—a dual-threaded run took one hour and twenty minutes. Clearly, running this for the entire book without suitably distributing the process would take far too long a period of time for the spell correction to be useful.

Spell Corrector is our distributed system to accomplish this goal. It implements a distributed master-slave system through the use of the threading and concurrency libraries packaged in C++11 standard, as well as an implementation of the MapReduce paradigm.

Utilizing this system, the same test set was corrected in just over fourteen minutes—over five times faster than the dual-threaded version.

2 Architecture of Components

Spell Corrector is built on top of `libdistributed`, a master-slave architecture that utilizes C++11's concurrency libraries. There are three major components to the Spell Corrector system: the Master, the Slave, and the Client.

The Master

The master performs load balancing for the system: when a client requests a slave, the master assigns a slave based on how much load is on the slave.

The Client

The client fulfills the job of a Mapper: it generates key-value pairs that will be passed to the Reducer for processing. The client takes in the output from Tesseract in the form of a text file. After connecting to the master, the client breaks the text file into individual jobs, composed of a certain number of lines. The exact number of lines in each job is determined by the filesize. After each job has been created, the client requests a slave from the master and sends a message to the slave.

This message contains all of the information the slave needs to do the work the job requires: the type of job the client is requesting, a key individual to the job, the number of lines in the job, and the job text itself. The type of job can be one of two options: either a request to do the job, or a request to return the result. The key given to the slave is designed to be unique to the job: it consists of the text file's name, a random 64-bit number unique to the client, the instantiation time, and the job's placement in the file. In the terminology of the MapReduce paradigm, the key-value pair would be the key and the job text.

After sending each job to its respective slave, the client waits for a confirmation message from the slave. If the slave responds that there was an error or that it timed out, the client resends the job. Otherwise, the client receives a "Job received" message, and waits for a small period of time for the slaves to work.

After waiting, the client sends a job to each slave in order, asking if the job has been finished yet. If the slave responds positively, the client receives the corrected text from the slave and writes it to an output file. If the slave responds negatively, the client waits for a second before asking again. If no response is received, the job is resent, and the client waits for the job to be finished.

The Slave

The slave fulfills the function of a Reducer, doing the spelling correction of the value of the key-value pair.

After the slave receives a job from the client, it checks what type of job was requested. If the job is a request to do work, the slave checks that the job is valid. If the job is indeed valid, the slave reports to the client that it received the message, and begins to work. If the job isn't valid, the slave reports an error. After the validity check, the slave writes the received message to a file, named after the key, and runs the spell correction on the saved file in a separate thread. The slave generates as many threads as there are available processors on the machine it is run on.

When the spell correction is done, the now corrected file is renamed. The filename is the string "processed-" concatenated with the key—this is for easy lookup. When the client requests the finished job, the slave simply looks to see if the processed file exists. If so, the slave returns the fully processed text; if not, the slave responds that it needs more time.

In the MapReduce paradigm's terminology, the slave fulfills the function of a Reducer: it takes in the key-value pairs generated by the Mapper (client) and performs its operation upon the value.

3 Algorithm

The function that Spell Corrector performs is spell correction based on a Bigram Language model. A unigram is a single word—certain words appear in language far more often than others, and consequently have a certain probability assigned to them. Similarly, a bigram is a sequence of two words. Spell Corrector has a database of the probabilities of both unigrams and bigrams, generated through a large set of training data by counting all of the unigrams and bigrams. Dividing the count by the total number of unigrams or bigrams approximates the probability. Correcting spelling through the use of unigrams is the most basic method—it is essentially spell correction looking at a single word t a time. The Bigram Language Model is a step up, looking not only at the probability that a single word is correct, but also that it makes sense given the context of the surrounding words. While this leads to more accurate results, it has the necessary tradeoff of being much more time- and memory-intensive. Due to the nature of bigrams, there can be as many bigrams beginning with a certain word as there are words.

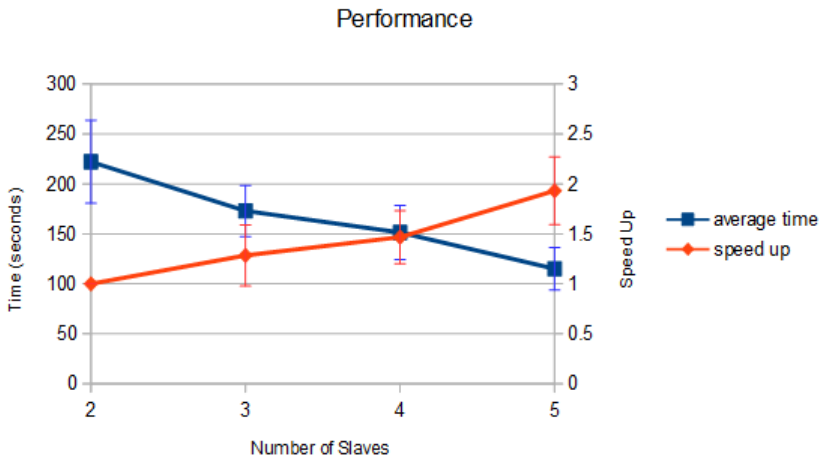
The Corrector runs a search that takes a single misspelled word and returns a list of words from the unigram dictionary that are likely to have been the intended word. Only words within a certain word length of the misspelled word are considered. The misspelled word is then compared to the candidate words, utilizing both the longest common substring of the words and also knowledge of the types of mistakes that Tesseract tends to make. This comparison generates a probability that the candidate word is the correct word; if this probbility is above a certain threshold, it is combined with the probability that the word will occur in an English text, and added to the list of returned words. This list is sorted by descending probability, so that the most likely word is first.

After this list has been generated, each term is then looked up in the bigram dictionary, utilizing the surrounding words. The probability associated with each bigram is combined with the preexisting probability, and the entry with the highest total probability is the most likely candidate for the misspelled word.

4 Results

Spell Corrector's purpose is to correct the output of the Tesseract OCR engine, and as such its accuracy is dependent upon the accuracy of Tesseract. The system was produced as part of Elizabeth Kilson's senior project, utilizing a small amount of multithreading. This implementation has proven to reduce the number of errors by approximately half [1]. Our distributed version outputs the same results as those in [1], and as such has the same accuracy.

In terms of speed, however, our version has major improvements. As stated earlier, testing on both versions showed a performance increase of 560% after distributing using two slaves. Further testing produced the curve shown in Figure 1.



Results/cloud2b.PNG

Figure 1: Performance Curve for Spell Corrector

5 References

[1] Kilson, Elizabeth and Yeo, James. *Blind Reading Assist Technology.* The Cooper Union for the Advancement of Science and Art, 6 May 2014. May 13, 2014.

[2] Miller, Claire Cain and Bosman, Julie. "Siding With Google, Judge Says Book Search Does Not Infringe Copywrite." *The New York Times* 14 Nov. 2013: Web May 13, 2014.

[3] Taycher, Leonid. "Books of the world, stand up and be counted! All 129,864,880 of you." *Google Books Search* 5 Aug. 2010 Web May 13, 2014.